Llenando un formulario HTML automáticamente con AJAX

Christian Velasco
29/10/2020

Durante el desarrollo de un proyecto web de una clínica, nuestro cliente solicitó que el registro de pacientes lo hagamos de forma automática a través de un archivo JSON que tiene el registro de clientes históricos de la clínica. ¿Cómo podemos hacer eso?

Una solución factible es adicionar un botón `botónBuscar` y que tenga como responsabilidad conectarse a un servidor externo (API) y realizar una importación de datos del archivo JSON.

Esta importación es realizada a través de una requisición AJAX con JavaScript. Para ello necesitamos de un objeto llamado `XMLHttpRequest`. Pero antes de continuar vamos a explicar lo que es AJAX.

Entendiendo AJAX

A través de Javascript conseguimos enviar solicitudes a determinadas URL y obtener sus respuestas sin que se recarguen las páginas, es decir, cuando el usuario haga clic en el botón `botónBuscar` del formulario, se realizará la búsqueda y se rellenarán las informaciones de los pacientes sin recargar la página.

Este método de solicitud en Javascript se conoce como AJAX. Genial, pero ¿cómo podríamos aplicar este AJAX?

Vamos a crear nuestra propia funcionalidad para buscar esos pacientes de otro servidor y así podremos tener el resultado deseado.

Obtener el formulario HTML

Nuestro primer paso será en realidad, obtener nuestro formulario HTML para que apliquemos el AJAX. Para este ejemplo usaré este formulario, que está en el Github de Alura, es decir, si quieres seguir el paso a paso, ¡también puedes descargarlo! ;)

<>)
Con el HTML listo, necesitamos comenzar a implementar la importación de los pacientes desde un servidor externo conforme lo solicitó nuestro cliente. Los datos de los pacientes se encuentran en una API de Git y para acceder a ellos vamos a usar solicitudes con `XMLHttpRequest`

Creando solicitudes con XMLHttpRequest

Javascript nativo tiene un objeto responsable por realizar solicitudes AJAX conocido como XMLHttpRequest, es decir, por medio de él podemos recuperar datos de una URL determinada. A pesar de su nombre, se puede utilizar para recuperar cualquier tipo de datos y brinda soporte a los protocolos HTTP, FILE y FTP.

Si observamos nuestro código HTML, el botón **Buscar Pacientes** ya vino creado. Y dentro del Script ya tenemos 3 funciones creadas: `adicionarPacienteEnLaTabla`, `construirTr` y `construirTd`. Estas funciones se encargarán de adicionar los pacientes importados a la tabla. Antes de crear nuestra solicitud vamos a seleccionar el botón con `document.querySelector` y colocamos un `eventListener` del evento clic dentro de él. Hacemos esto al final de la función `construirTd`.

```
var botonBuscar = document.querySelector("#buscar-paciente");
botonBuscar.addEventListener("click",function(){
};
```

Ahora sí, dentro del evento vamos a crear una instancia del objeto para comenzar a usarlo.

```
var botonBuscar = document.querySelector("#buscar-paciente");
botonBuscar.addEventListener("click",function(){
```

```
    var  xhr = new XMLHttpRequest();
};
```

Ahora el `XMLHttpRequest` necesita ser configurado, vamos a definir cuál método queremos utilizar en la requisición y para cual servidor vamos a enviarla. Para configurar el `XMLHttpRequest` usamos la función `.open()`

El método open recibe tres parámetros, los cuales son:

1. El verbo HTTP que se usará para realizar la solicitud, y debe seguir el estándar REST.
2. La URL que pretendemos obtener los datos.
3. Un argumento booleano que indica si la solicitud debe ser asíncrona o síncrona (por defecto es igual a asíncrono, valor booleano = `true`)

```
var botonBuscar = document.querySelector("#buscar-paciente");
botonBuscar.addEventListener("click",function(){
    var  xhr = new XMLHttpRequest();
    xhr.open("GET","https://alura-es-cursos.github.io/api-pacientes/pacientes.
};
```

Por último, para enviar la requisición necesitamos llamar al método send():

```
var botonBuscar = document.querySelector("#buscar-paciente");
botonBuscar.addEventListener("click",function(){
    var  xhr = new XMLHttpRequest();
    xhr.open("GET","https://alura-es-cursos.github.io/api-pacientes/pacientes.
    xhr.send;
};
```

Si en este momento realizamos clic en el botón, la requisición será enviada, sin embargo, no veremos ningún resultado, pues no estamos capturando la respuesta.

Capturando la respuesta del XMLHttpRequest

Para capturar la respuesta cuando la requisición HTTP es retornada, necesitamos colocar un escuchador de eventos en el propio `XMLHttpRequest`, capturando al evento `load`.

```
var botonBuscar = document.querySelector("#buscar-paciente");
botonBuscar.addEventListener("click",function(){
        var xhr = new XMLHttpRequest;
        xhr.open("GET","https://alura-es-cursos.github.io/api-pacientes/pacient
         xhr.addEventListener("load", function() {
                 console.log(xhr.responseText);
         });
         xhr.send();
};
```

Ahora si realizamos clic en el botón y vemos en la consola, veremos los datos importados.

Transformado los datos de JSON para un objeto de JavaScript

Si observamos en la consola lo que está siendo impreso, veremos que nos retorna un archivo JSON, que es un formato común para transitar datos en la web. Como no queremos trabajar con texto, vamos a transformar ese texto en un objeto de `JavaScript`, para ello usamos la función `JSON.parse()`.

```
var botonBuscar = document.querySelector("#buscar-paciente");
botonBuscar.addEventListener("click",function(){
        var xhr = new XMLHttpRequest;
        xhr.open("GET","https://alura-es-cursos.github.io/api-pacientes/pacien
        xhr.addEventListener("load", function() {
              var respuesta = xhr.responseText;
              var pacientes = JSON.parse(respuesta);
        });
```

```
        xhr.send();
};
```

Adicionando los pacientes en la tabla

En nuestro código ya tenemos las funciones que se encargan de adicionar pacientes en la tabla, vamos a aprovecharla para recorrer el array de pacientes y adicionar cada uno de ellos:

```
var botonBuscar = document.querySelector("#buscar-paciente");
botonBuscar.addEventListener("click",function(){
        var xhr = new XMLHttpRequest;
        xhr.open("GET","https://alura-es-cursos.github.io/api-pacientes/pacien
        xhr.addEventListener("load", function() {
              var respuesta = xhr.responseText;
              var pacientes = JSON.parse(respuesta);
              //adicionar aquí
                        pacientes.forEach(function(paciente){
              adicionarPacienteEnLaTabla(paciente);
               });
        });
        xhr.send();
};
```

Si hacemos clic en el botón **Buscar Pacientes** veremos que los pacientes están siendo añadidos en la tabla.

<>)

Lidiando con errores

Nuestra requisición AJAX ya está implementada correctamente, solo que no estamos considerando que existe la posibilidad de que se presenten errores durante la requisición, por lo tanto, vamos a detectar los problemas y exhibir un mensaje de error para informar al

usuario. Para detectar si hubo un error, debemos utilizar el código de status de la requisición HTTP, que puede ser obtenido a través de la propiedad `status` del `XMLHttpRequest`. Vamos a crear una lógica que verifique si el código del `status` es `200`, que significa que la requisición fue exitosa, y caso contrario, vamos a exhibir los errores para el usuario.

```
var botonBuscar = document.querySelector("#buscar-paciente");
botonBuscar.addEventListener("click",function(){
        var xhr = new XMLHttpRequest;
        xhr.open("GET","https://alura-es-cursos.github.io/api-pacientes/pacien
        xhr.addEventListener("load", function() {

       if(xhr.status == 200){
                var respuesta = xhr.responseText;
                var pacientes = JSON.parse(respuesta);
                pacientes.forEach(function(paciente){
              adicionarPacienteEnLaTabla(paciente);
                });
       } else {
//vamos a exhibir los errores aquí
              }
        });
        xhr.send();
};
```

Caso la requisición muestre cualquier otro código que no sea 200, debemos exhibir un pequeño mensaje de error al usuario. Este mensaje será una etiqueta `<span>` que la colocaremos en nuestro HTML antes del botón `buscar-paciente`.

```
<span id="error-ajax">Error al buscar los pacientes</span>
```

Luego vamos a alterar el estado del botón a través de un selector con JavaScript, iniciando el mismo con la propiedad `visibility = "hidden"`, o sea oculto. Este bloque lo colocamos antes del `querySelector` del botón `buscar-paciente`

```
var errorAjax = document.querySelector("#error-ajax");
errorAjax.style.visibility = "hidden";
```

Para finaliza en el `else` de nuestro bloque `if` alteramos la propiedad `visibility` del objeto `error-ajax` para `"visible"`.

```
var botonBuscar = document.querySelector("#buscar-paciente");
botonBuscar.addEventListener("click",function(){
        var xhr = new XMLHttpRequest;
        xhr.open("GET","https://alura-es-cursos.github.io/api-pacientes/pacien
        xhr.addEventListener("load", function() {

       if(xhr.status == 200){
                var respuesta = xhr.responseText;
                var pacientes = JSON.parse(respuesta);
                pacientes.forEach(function(paciente){
              adicionarPacienteEnLaTabla(paciente);
                });
       } else {
errorAjax.style.visibility = "visible"
              }
        });
        xhr.send();
};
```

Si simulamos un error, por ejemplo, removiendo una parte de la URL del servidor, verificamos si el mensaje de error de la etiqueta `<span>` va a aparecer.

<>)
(Aquí)[https://raw.githubusercontent.com/alura-es-cursos/api-pacientes/artigo-ajax-javascript-full/conexion-ajax-clinica-vida-y-salud-full.html] puedes descargar el código completo.

¿Qué tal aprender más sobre **JavaScript** y sus diversos recursos? Entonces, ¡Mira nuestros cursos aquí en Alura!

ARTÍCULOS DE TECNOLOGÍA > FRONT END

CURSOS

Cursos de Programación
Lógica de Programación | Java

Cursos de Front End
HTML y CSS | JavaScript | React

Cursos de Data Science
Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

Cursos de DevOps
Docker | Linux

Cursos de Innovación y Gestión
Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics | Liderazgo y Gestión de Equipos | Startups y Emprendimiento